

901141

2 of 3

MEMORANDUM

RM-4137-PR

JUNE 1964

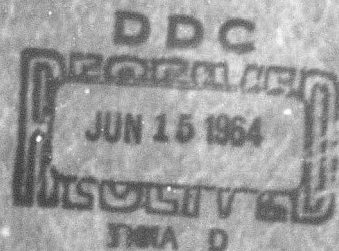
35-P \$1.00

SCHEDULING IN PROJECT NETWORKS

D. R. Fulkerson

PREPARED FOR:

UNITED STATES AIR FORCE PROJECT RAND



The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM

RM-4137-PR

JUNE 1964

SCHEDULING IN PROJECT NETWORKS

D. R. Fulkerson

This research is sponsored by the United States Air Force under Project RAND—contract No. AF 49(638)-700 monitored by the Directorate of Development Planning, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

DDC AVAILABILITY NOTICE

Qualified requesters may obtain copies of this report from the Defense Documentation Center (DDC).

The **RAND** *Corporation*

1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

PREFACE

Project scheduling using networks has become increasingly popular in the last few years. This Memorandum discusses three scheduling problems of this variety.

This paper will appear in the Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems.

SUMMARY

Problems that involve a schedule or timetable of projected operations or jobs occur frequently in operations research. Three such scheduling problems are discussed in this survey. Each of these problems poses a question about a finite partially ordered set of jobs, a question that can, in each case, be reformulated in terms of flows in acyclic directed networks. Solution procedures for each problem are described.

SCHEDULING IN PROJECT NETWORKS

1. INTRODUCTION

Problems that involve a schedule or timetable of projected operations or jobs occur very frequently in operations research. Such scheduling problems usually involve optimization in some form or another, and, more often than not, turn out to have the unpleasant features of being both combinatorially huge and seemingly lacking in structure. On the one hand, the combinatorial magnitude of the problem precludes exhaustive enumerative methods of solution, while on the other hand, the lack of structure makes useful analysis difficult. Faced with this situation, the operations researcher frequently resorts to "solution" by computer simulation, rules-of-thumb, incomplete enumerative schemes, heuristic computer programs, or like devices which distress the mathematician. But the three scheduling problems discussed in this survey do not fall in this category of "bad" problems. Each of these problems poses a question about a finite partially ordered set of jobs, a question that can, in each case, be rephrased in terms of flows in networks. Consequently there are good algorithms available for solving these problems.

Section 2 below reviews relevant material concerning flows in networks [9]. The specific scheduling problems are then discussed in Secs. 3, 4, and 5. The problem of Sec. 3 was proposed several years ago by Tompkins [20].

A method of solution (though not the one described in Sec. 3) was later given by Dantzig and Fulkerson [2]. The problems of Secs. 4 and 5 have been written about extensively in operations research journals, newspapers, and popular magazines, and are frequently identified by such names as PERT (Program Evaluation and Review Technique), CPM (Critical Path Method), and others too numerous to list. The basic model of Sec. 4 was formulated and studied by Malcolm, Roseboom, Clark, and Fazar [18], and independently by Kelley and Walker [15], as a means of scheduling large, complicated projects composed of many individual jobs, each of which has a known duration time. The problem of Sec. 5 was formulated by Kelley and Walker [15]; it deals with the same basic model, but introduces further complications concerning cost-time relations for the jobs. Network flow methods of solution for this class of problems have been given by Kelley [16] and Fulkerson [10]. A related solution method, using longest chains, is described in Sec. 5. These models have had widespread industrial impact in the last few years, and are currently in extensive use.

2. FLOWS IN NETWORKS

A directed network (graph) $G = [N; \mathcal{Q}]$ consists of a finite collection N of elements $1, 2, \dots, n$, together with a subset \mathcal{Q} of the ordered pairs (i, j) of distinct elements of N . The elements of N will be called nodes; members of \mathcal{Q} are arcs. Figure 2.1 shows a directed network having

four nodes and six arcs $(1,2)$, $(1,3)$, $(2,3)$, $(2,4)$, $(3,2)$, and $(3,4)$.

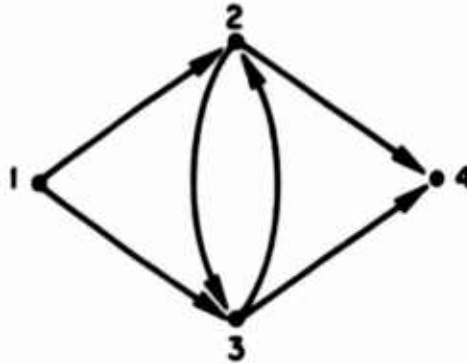


Fig. 2.1

Suppose that each arc (i,j) of a directed network has associated with it a nonnegative number c_{ij} , the capacity of (i,j) , which may be thought of as representing the maximal amount of some commodity that can arrive at j from i along (i,j) per unit time in a steady-state situation. Then a natural question is: What is the maximal amount of commodity flow from some node to another via the entire network? (For example, one might think of a network of city streets, the commodity being cars, and ask for a maximal traffic flow from some point to another.) We may formulate the question mathematically as follows. Let l and n be the two nodes in question. A flow of amount v , from l to n in $G = [N; \mathcal{A}]$ is a function x from \mathcal{A} to real numbers

(a vector x having components x_{ij} for (i,j) in \mathcal{Q}) that satisfies the linear equations and inequalities

$$(2.1) \quad \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} v, & i = 1, \\ -v, & i = n, \\ 0, & \text{otherwise,} \end{cases}$$

$$(2.2) \quad 0 \leq x_{ij} \leq c_{ij}, \quad (i,j) \text{ in } \mathcal{Q}.$$

In (2.1) the sums are of course over those nodes for which x is defined. We call 1 the source, n the sink. A maximal flow from source to sink is one that maximizes the variable v subject to (2.1), (2.2).

Figure 2.2 shows a flow from source node 1 to sink node 6 of amount 7. In Fig. 2.2, the first number of each pair beside an arc is the arc capacity, the second number the arc flow.

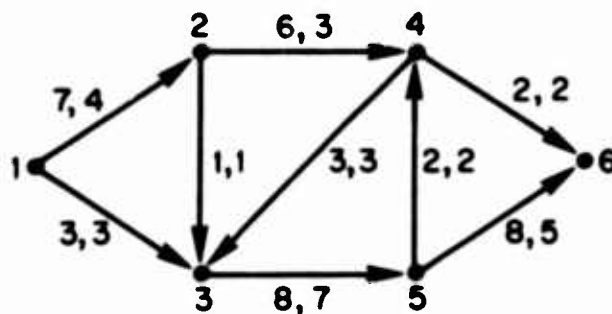


Fig. 2.2

To state the fundamental theorem about maximal flow, we need one other notion, that of a cut. A cut separating l and n is a partition of the nodes into two complementary sets, I and J, with l in I, say, and n in J. The capacity of the cut is then

$$(2.3) \quad \sum_{\substack{i \text{ in } I \\ j \text{ in } J}} c_{ij} .$$

(For instance, if $I = \{1,3,4\}$ in Fig. 2.2, the cut has capacity $c_{12} + c_{35} + c_{46} = 17$.) A cut separating source and sink of minimum capacity is a minimal cut, relative to the given source and sink.

Summing the equations (2.1) over i in the source-set I of a cut and using (2.2), we see that

$$(2.4) \quad v = \sum_{\substack{i \text{ in } I \\ j \text{ in } J}} (x_{ij} - x_{ji}) \leq \sum_{\substack{i \text{ in } I \\ j \text{ in } J}} c_{ij} .$$

In words, for an arbitrary flow and arbitrary cut, the net flow across the cut is the flow amount v , which is consequently bounded above by the cut capacity. Theorem 2.1 below asserts that equality holds in (2.4) for some flow and some cut, and hence the flow is maximal, the cut minimal [6].

Theorem 2.1. For any network the maximal flow amount from source to sink is equal to the minimal cut capacity relative to the source and sink.

Theorem 2.1 is a kind of combinatorial counterpart, for the special case of the maximal flow problem, of the duality theorem for linear programs, and can be deduced from it [3]. But the most revealing proof of Theorem 2.1 uses a simple "marking" or "labeling" process [7] for constructing a maximal flow, which also yields the following theorem.

Theorem 2.2. A flow x from source to sink is maximal if and only if there is no flow augmenting path with respect to x .

Here we need to say what an x -augmenting path is. First of all, a path from one node to another is a sequence of distinct end-to-end arcs that starts at the first node and terminates at the second; arcs traversed with their direction in going along the path are forward arcs of the path, while arcs traversed against their direction are reverse arcs of the path. A path from source to sink is x -augmenting provided that $x < c$ on forward arcs and $x > 0$ on reverse arcs. For example, the path (1,2), (2,4), (5,4), (5,6) in Fig. 2.2 is an augmenting path for the flow shown there. Figure 2.3 below indicates how such a path can be used to increase the amount of flow from source to sink.

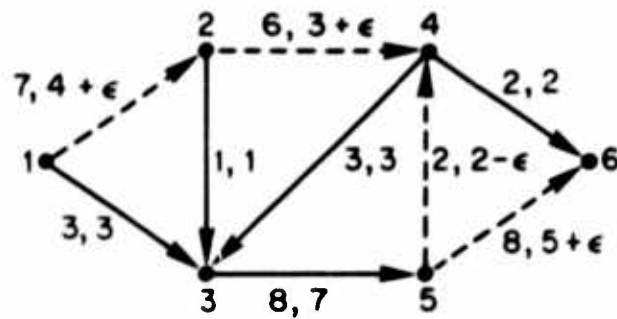


Fig. 2.3

Taking the flow change ϵ along the path as large as possible in Fig. 2.3, namely $\epsilon = 2$, produces a maximal flow, since the cut $I = \{1, 2, 4\}$, $J = \{3, 5, 6\}$ is then "saturated."

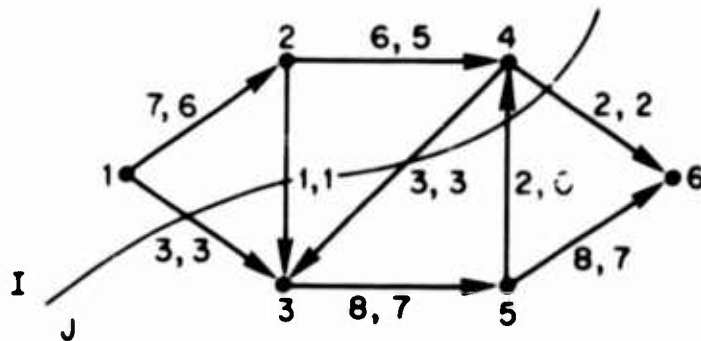


Fig. 2.4

The labeling process of [7] is a systematic and efficient search, fanning out from the source, for a flow augmenting path. If none such exists, the process ends by locating a minimal cut.

The following theorem, of special significance for combinatorial applications, is also a consequence of the procedure sketched above for constructing maximal flow.

Theorem 2.3. If all arc capacities are integers, there is an integral maximal flow.

It is sometimes convenient to alter the constraints (2.2) of the maximal flow problem to

$$(2.5) \quad \ell_{ij} \leq x_{ij} \leq c_{ij}.$$

Here ℓ is a given lower bound function satisfying $\ell \leq c$.

The analogue of Theorem 2.1 is then

Theorem 2.4. If there is a function x satisfying (2.1) and (2.5) for some number v , then the maximum v subject to these constraints is equal to the minimum of

$$(2.6) \quad \sum_{\substack{i \text{ in } I \\ j \text{ in } J}} (c_{ij} - \ell_{ji})$$

taken over all cuts I, J separating source and sink. On the other hand, the minimum v is equal to the maximum of

$$(2.7) \quad \sum_{\substack{i \text{ in } I \\ j \text{ in } J}} (\ell_{ij} - c_{ji})$$

taken over all cuts I, J separating source and sink.

Appropriate analogues of Theorem 2.2 are also valid for the construction of maximal or minimal flows satisfying lower and upper bounds on arcs. Hence if all l_{ij} and c_{ij} are integral, there exist integral maximal and minimal flows, provided feasible flows exist.

One of the most practical problem areas involving network flows is that of constructing flows satisfying constraints of various kinds and minimizing cost. The standard linear programming transportation problem, which has an extensive literature, is in this category.

We put the problem as follows. Each arc (i,j) of a network $G = [N; \mathcal{A}]$ has a capacity c_{ij} and a cost a_{ij} . It is desired to construct a flow x from source to sink of specified amount v that minimizes the total flow cost

$$(2.8) \quad \sum_{(i,j) \in \mathcal{A}} a_{ij} x_{ij}$$

over all flows that send v units from source to sink. In many applications one has supplies of a commodity at certain points in a transportation network, demands at others, and the objective is to satisfy the demands from the supplies at minimum cost. It is easy to convert such a problem to the form described above.

By treating v as a parameter, the method for constructing maximal flows can be used to construct minimal cost flows throughout the feasible range of v . Indeed, the

solution procedure can be viewed as one of solving a sequence of maximal flow problems, each on a subnetwork of the original one [8]. Another, not essentially different, viewpoint is provided by the following theorem [1, 13].

Theorem 2.5. Let x be a minimal cost flow from source to sink of amount v . Then the flow obtained from x by adding $\epsilon > 0$ to the flow in forward arcs of a minimal cost x -augmenting path, and subtracting ϵ from the flow in reverse arcs of this path, is a minimal cost flow of amount $v + \epsilon$.

Here the cost of a path is the sum of arc costs over forward arcs minus the corresponding sum over reverse arcs, i.e., the cost of "sending an additional unit" via the path.

Thus, if all arc costs a_{ij} are nonnegative, for example, one can start with the zero flow and apply Theorem 2.5 to obtain minimal cost flows for increasing v . (The cost profile thereby generated is piecewise linear and convex.) All that is needed to make this an explicit algorithm is a method of searching for a minimal cost flow augmenting path. Various ways of doing this can be described. One such will be given below.

These methods produce integral flows in case the arc capacities (and lower bounds) are integers. Theoretical upper bounds on the computing task, ones that are quite good, are easily obtained in each case. This may be contrasted with the situation for general linear programs,

where decent upper bounds on solution methods are unknown.

In order to describe a procedure for locating minimum cost flow augmenting paths, we begin with the following problem. Consider a directed network in which each arc (i,j) has associated with it a positive number a_{ij} , which may be thought of as the length of the arc, or the cost of traversing the arc. How does one determine a shortest chain from some node to another? Here we have used chain to mean a path containing only forward arcs, the length of the chain being obtained by adding its arc lengths.

Many ways of locating shortest chains efficiently have been suggested. We describe one [5]. Like others, it simultaneously finds shortest chains from the first node to all others reachable by chains.

In this method each node i will initially be assigned a number π_i . These node numbers, which we shall refer to as potentials, will then be revised in an iterative fashion. Let 1 be the first node. To start, take $\pi_1 = 0$, $\pi_i = \infty$ for $i \neq 1$. Then search the list of arcs for an arc (i,j) whose end potentials satisfy

$$(2.9) \quad \pi_i + a_{ij} < \pi_j .$$

(Here $\infty + a = \infty$). If such an arc is found, change π_j to $\pi_j' = \pi_i + a_{ij}$, and search again for an arc satisfying (2.9), using the new node potentials. Stop the process when the

node potentials satisfy

$$(2.10) \quad \pi_i + a_{ij} \geq \pi_j$$

for all arcs.

It is not hard to show that the process terminates, and that when this happens, the potential π_j is the length of a shortest chain from 1 to j. (Here $\pi_j = \infty$ at termination means there is no chain from 1 to j.) A shortest chain from 1 to j can be found by tracing back from j to 1 along arcs satisfying (2.10) with equality (see Fig. 2.5).

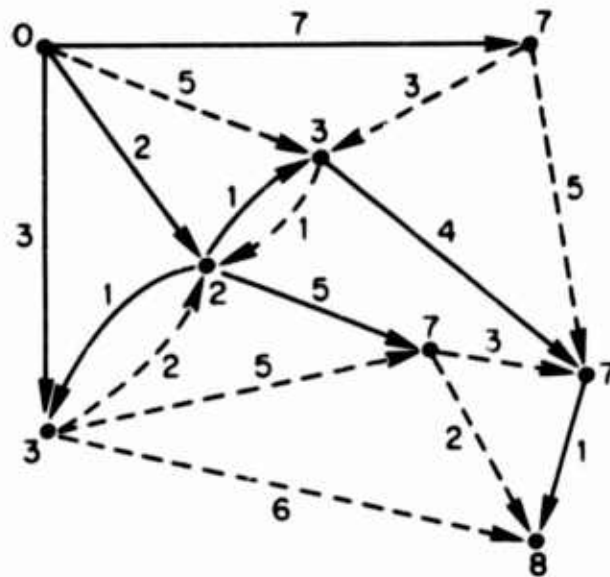


Fig. 2.5

While we have assumed positive lengths for the method described above, this assumption can be weakened. Call a chain of arcs leading from a node to itself a directed cycle. Then it is enough to suppose that all directed cycle lengths are nonnegative.*

If directed cycle costs are nonnegative, the minimum cost flow problem can be solved by repeatedly finding cheapest chains in suitable networks. Because of the assumption on the cost function a , we may start with the zero flow. Thus, using Theorem 2.5, it is enough to reduce the problem of finding a cheapest flow augmenting path with respect to a minimal cost flow x of amount v to that of finding a cheapest chain. Define a new network $G' = [N; \mathcal{Q}']$ from the given one $G = [N; \mathcal{Q}]$ and the flow x as follows. First note that we may assume $x_{ij} \cdot x_{ji} = 0$, since $a_{ij} + a_{ji} \geq 0$. Now put (i, j) in \mathcal{Q}' if either $x_{ij} < c_{ij}$ or $x_{ji} > 0$, and define a' by

$$(2.11) \quad a'_{ij} = \begin{cases} a_{ij} & \text{if } x_{ij} < c_{ij} \text{ and } x_{ji} = 0, \\ -a_{ji} & \text{if } x_{ji} > 0. \end{cases}$$

* This assumption appears essential in the sense that the problem of finding a shortest (simple) chain from one node to another in a network whose arcs may have arbitrary lengths can be shown to be equivalent to the traveling salesman problem, for which no simple methods are known.

Thus a chain from source to sink in the new network corresponds to an x -augmenting path in the old, and these have the same cost. Moreover, since x is a minimal cost flow, the function a' satisfies the nonnegative directed cycle condition. Hence the method described above can be used to construct minimal cost flows of successively larger amounts.

If the network is acyclic (contains no directed cycles), the shortest chain method can be modified in such a way that once a potential is assigned a node, it remains unchanged. One can begin by numbering the nodes so that if (i,j) is an arc, then $i < j$. Such a numbering can be obtained as follows. Since the network is acyclic, there are nodes having no inward-pointing arcs. Number these nodes 1, 2, ..., k in any order. Next delete these nodes and all their arcs, search the new network for nodes having no inward-pointing arcs, and number these, starting with $k+1$. Repetition of this process leads to the desired kind of numbering (see Fig. 2.6).

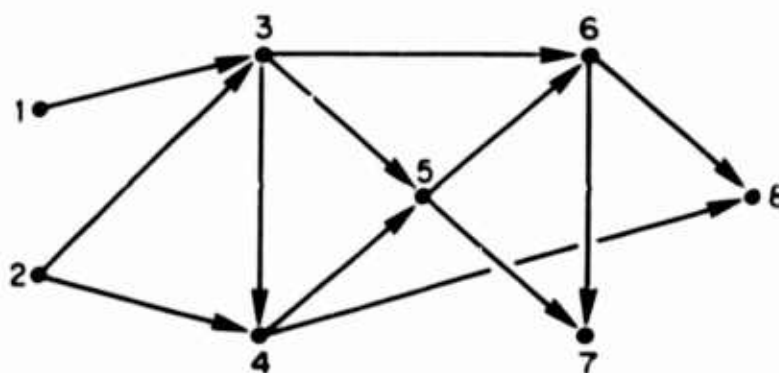


Fig. 2.6

If we wish to find shortest chains from node k to all other nodes reachable from k by chains, the calculation is now trivial. Simply define $\pi_k, \pi_{k+1}, \dots, \pi_n$ recursively by

$$(2.12) \quad \begin{cases} \pi_k = 0, \\ \dots \\ \pi_j = \min_{k \leq i < j} (\pi_i + a_{ij}), \quad j = k+1, \dots, n. \end{cases}$$

Here the minimum is of course taken over i such that (i, j) is an arc.

Longest chains in acyclic networks can be computed by replacing "min" by "max" in (2.12).

3. MINIMUM NUMBER OF MACHINES TO MEET A FIXED JOB SCHEDULE

Suppose there are n jobs $1, 2, \dots, n$ with specified start and finish times a_1, a_2, \dots, a_n and b_1, b_2, \dots, b_n , with $a_i \leq b_i$. In other words, the schedule of starting times for the various jobs is fixed in advance, and the duration times $t_i = b_i - a_i$ of the jobs are known. Assume that we have a number of (identical) machines, each of which can perform any job in the specified time, and that the reassignment or set-up time required for a machine to go from job i to job j is $r_{ij} \geq 0, i, j = 1, 2, \dots, n$. What is the minimum number of machines required to meet the given job schedule? For a concrete example, think of an airline, say, which wants to meet a fixed flight schedule with the

minimum number of planes, all of the same type. Start and finish times are known for each flight, and the times r_{ij} to return from the destination point of flight i to the origin point of flight j are also known.

Making the reasonable assumption that the reassignment times satisfy

$$(3.1) \quad r_{ij} \leq r_{ik} + r_{kj}$$

for all i, j, k , it is easy to check that the jobs can be partially ordered by saying that i precedes j if

$$(3.2) \quad b_i + r_{ij} \leq a_j .$$

We may depict the order relations among the jobs by means of an acyclic directed network whose arcs represent jobs. To take a simple case, suppose there are five jobs with the ordering: 1 precedes 3, 1 and 2 precede 4, and 1, 2, 3, 4 precede 5. This may be pictured by the network shown in Fig. 3.1 below.

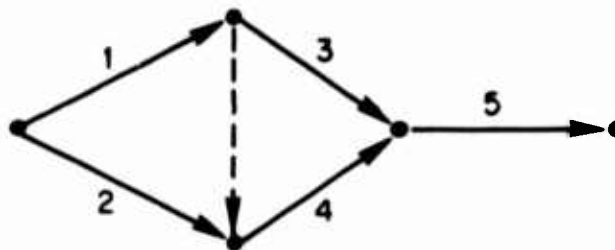


Fig. 3.1

Notice that we have added a "dummy" job, the dotted arc of Fig. 3.1, to maintain the proper order relations among the jobs. It is not difficult to show that the use of dummies permits a network representation of this kind for any finite partially ordered set.

Since a chain of arcs in this network represents a possible assignment of jobs to one machine, the problem is to cover all non-dummy arcs with the minimum number of chains. Using the integrity theorem, this can be made into a flow problem as follows. Add a node to the network, the source for flow, and direct dummy arcs from this node to all nodes of the network that have only outward-pointing arcs. Similarly add a sink node, directing dummy arcs into this from all nodes having only inward-pointing arcs. Now place a lower bound of 1 on each non-dummy arc, a lower bound of 0 on each dummy arc, and take all arc capacities infinite. Then an integral flow through the enlarged network of amount v picks out v chains (not necessarily distinct) that cover all non-dummy arcs, and consequently we wish to minimize v subject to (2.1) and (2.5). This can be done by a suitable labeling process which locates flow decreasing paths.

It can also be seen that the second half of Theorem 2.4 implies the following theorem for acyclic directed networks. This theorem is closely related to a theorem of Dilworth on chain decompositions of partially ordered sets [4].

Theorem 3.1. The minimum number of chains in an acyclic directed network required to cover a subset of arcs is equal to the maximum number of arcs of the subset having the property that no two belong to any chain.

In terms of the job scheduling problem, Theorem 3.1 asserts that the minimum number of machines required is equal to the maximum number of jobs, no two of which can be done by one machine. For example, in the network of Fig. 3.2 below, three chains are required to cover the solid arcs (as indicated by the flow shown in the figure), and jobs 2, 3, 4, for example, constitute a maximal set of jobs, no two of which can be done by one machine.

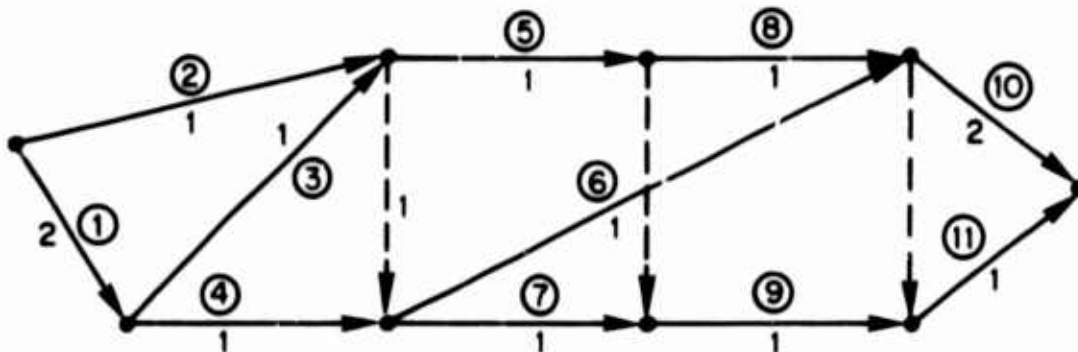


Fig. 3.2

Problems of this nature become considerably more complicated if the assumption of a fixed schedule is dropped. For instance, suppose the times a_i and b_i are at our disposal subject to the restrictions that $b_i - a_i \geq t_i$, with the duration times t_i known, as well as the reassignment times r_{ij} . The problem might then be to arrange a schedule which finishes all jobs by a given time and requires the minimum number of machines, or to finish all jobs at the earliest possible time with a fixed number of machines. For such scheduling problems there is very little known in the way of general theoretical results or good computational procedures. However, some special results have been deduced, notably by Johnson [14] and by Hu [12].

The problem of this section can also be viewed in terms of matrices of zeros and ones [9]. For instance, we may form an n by n (0,1)-matrix $A = (a_{ij})$ by setting $a_{ij} = 1$ if job i precedes job j , and $a_{ij} = 0$ otherwise. If we let $\rho(A)$ denote the term rank of A (the maximum number of 1's of A such that no two lie in the same row or column [19]), then it can be shown that the minimum number of machines required is equal to $n - \rho(A)$. Since the calculation of term rank can also be posed as a flow problem, this provides another flow formulation of the minimum machine scheduling problem.

4. PROJECT SCHEDULING

As noted in Sec. 1, one of the most popular combinatorial applications involving networks deals with the planning and scheduling of large complicated projects. Suppose that such a project (the construction of a bridge, for example) is broken down into many hundreds or thousands of individual jobs. Certain of these jobs will have to be finished before others can be started. Again we depict the partial ordering of jobs by an acyclic directed network, some of whose arcs correspond to actual jobs, as in Sec. 3.

Assuming that each job has a known duration time (dummies have zero duration times), and that the only scheduling restriction is that all inward-pointing jobs at a node must be finished before any outward-pointing job can be started, it follows that the minimum time to complete the entire project is equal to the length of a longest chain of jobs. Hence the minimum project time can be calculated easily by the recursive method described at the end of Sec. 2.

Figure 4.1 below provides an example of such a calculation. The number recorded beside each arc is the job duration time, and the number beside a node is the length of a longest chain from the starting node to the node in question.

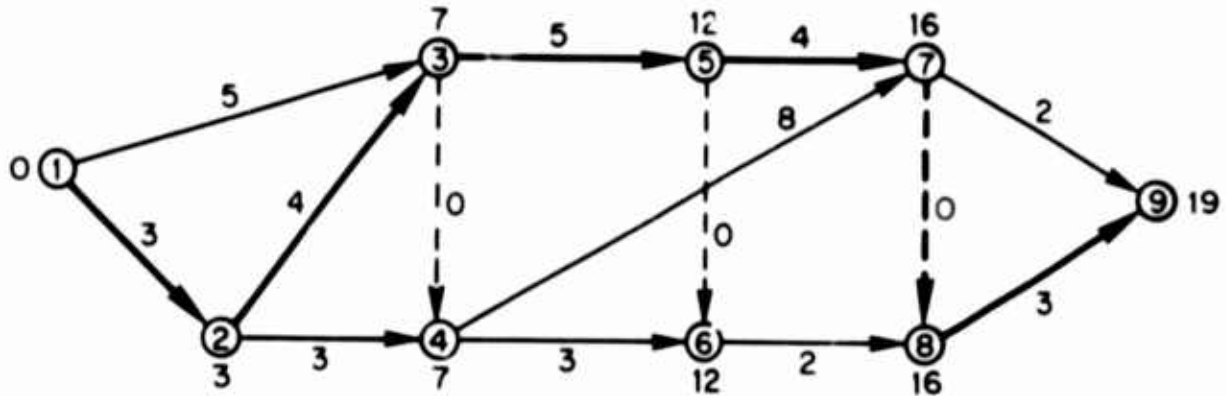


Fig. 4.1

The heavy arcs in Fig. 4.1 pick out a longest chain from node 1 to node 9. Such a chain is called critical, and the jobs which constitute a critical chain are called critical jobs. Some critical job must be expedited if total project time is to be shortened. The nodes of the project network are usually called events, and the times recorded by them are event times. For example, node 7 is the "event" of finishing its inward-pointing jobs, which event can occur at time 16. The event times provide a schedule for all jobs in the project.

The PERT model of a project usually assumes independent random variables for job times, instead of deterministic times as we have assumed above. But the usual practice has been to replace these random variables by their expected values, thereby obtaining a deterministic problem. The

solution of this deterministic problem always provides an optimistic estimate of the expected length of the project. One method for computing a better lower bound on expected project time has been given in [11]. It appears difficult to obtain very explicit information about the distribution of project duration time from known distributions of job times. In a practical situation, if such information is deemed desirable, one can always Monte Carlo the project on a computer. We refer the reader to [17, 21] for a fuller discussion of this.

Although the analysis of a PERT model, with fixed job times, is trivial from the mathematical point of view, the model itself appears to be a useful one, judging from its widespread acceptance and use throughout industry today. But it should be added that it is difficult to assess the usefulness of PERT on this basis alone, since the model has been the subject of much hard-sell advertising and exaggerated claims.

5. PROJECT COST CURVES

The PERT model can be complicated in various ways. One of the more interesting of these is to assume that a job can be expedited by spending more money on it, thereby raising the question: Which jobs should money be spent on, and how much, in order that the project be finished by a given date at minimum cost? In this section we shall assume that the time-cost relation for each job (i,j) is

linear. Specifically, we suppose that each arc (i,j) of the project network has associated with it three nonnegative integers,

$$(5.1) \quad a_{ij}, b_{ij}, c_{ij}$$

with $a_{ij} \leq b_{ij}$, the interpretation being that a_{ij} is the crash time for (i,j) , b_{ij} the normal completion time, while c_{ij} is the decrease in cost of doing (i,j) per unit increase in time from a_{ij} to b_{ij} . In other words, the cost of doing (i,j) in t_{ij} units of time is given by the known linear function

$$(5.2) \quad k_{ij} - c_{ij} t_{ij}$$

over the interval

$$(5.3) \quad a_{ij} \leq t_{ij} \leq b_{ij} .$$

Then, given λ units of time in which to finish the project, the problem is to choose job times t_{ij} and event times t_i satisfying

$$(5.4) \quad \begin{aligned} t_{ij} + t_i - t_j &\leq 0 , \\ t_n - t_1 &\leq \lambda , \\ a_{ij} &\leq t_{ij} \leq b_{ij} , \end{aligned}$$

and maximizing

$$(5.5) \quad \sum_{ij} c_{ij} t_{ij} .$$

For dummy jobs we may take $a_{ij} = b_{ij} = c_{ij} = 0$ in this linear program. We may also assume $t_1 = 0$, of course.

After some simplification, the dual of this linear program can be phrased as the following network flow problem. Find nonnegative numbers x_{ij} , one for each arc of the project network, and a nonnegative v , which satisfy the flow constraints (2.1) and minimize the nonlinear function

$$(5.6) \quad \lambda v + \sum_{ij} [b_{ij} \max(0, c_{ij} - x_{ij}) - a_{ij} \max(0, x_{ij} - c_{ij})] .$$

The function in brackets in (5.6) is sketched in Fig. 5.1. Since it is piecewise linear and convex, the theory outlined in Sec. 2 can be applied. That is, the program (2.1), (5.6) can be solved parametrically in v (and the dual problem (5.4), (5.5) parametrically in λ) either as a sequence of maximal flow problems or as a sequence of extremal chain problems in appropriate networks. Maximal flow approaches have been described in [10, 16]. Here we shall sketch a longest chain approach, using a small numerical example for illustration.

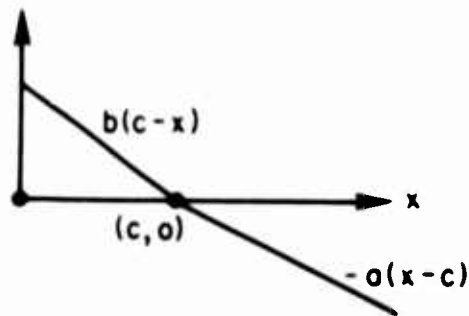


Fig. 5.1

Consider the project network and job data shown in Fig. 5.2, where we have recorded the data for arc (i,j) in the order b_{ij}, c_{ij}, a_{ij} . The first step in the computational procedure is to take job times at their upper bounds b_{ij} and find the corresponding event times t_i and critical chain, as shown in Fig. 5.3. This constitutes an optimal solution for $\lambda = t_4 = 11$.

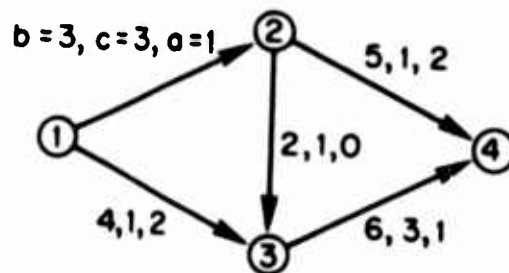


Fig. 5.2

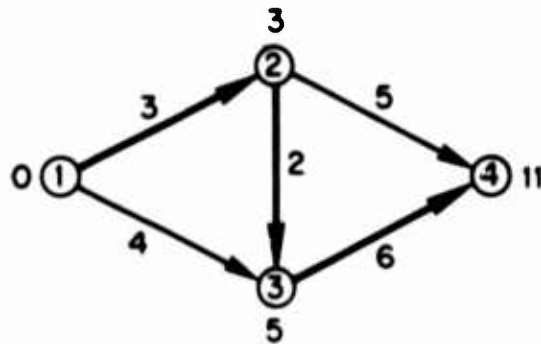


Fig. 5.3

We next impose flow along the critical chain, treating the c_{ij} as arc capacities at this step. Here we take a flow of one unit, the bind occurring on arc (2,3). Using the resulting flow and the problem data, we now form a new network. Corresponding to arc (i,j) of the project network, we consider the following possibilities:

- (a) $x_{ij} = 0$,
- (b) $0 < x_{ij} < c_{ij}$,
- (c) $x_{ij} = c_{ij}$,
- (d) $c_{ij} < x_{ij}$.

In case (a), put in an arc (i,j) with length b_{ij} ; in case (b), put in an arc (i,j) with length b_{ij} and an arc (j,i) with length $-b_{ij}$; in case (c), put in an arc (i,j) with length a_{ij} and an arc (j,i) with length $-b_{ij}$; in case (d), put in an arc (i,j) with length a_{ij} and an arc (j,i) with

length $-a_{ij}$. For the example we have the network shown in Fig. 5.4. Although this network contains

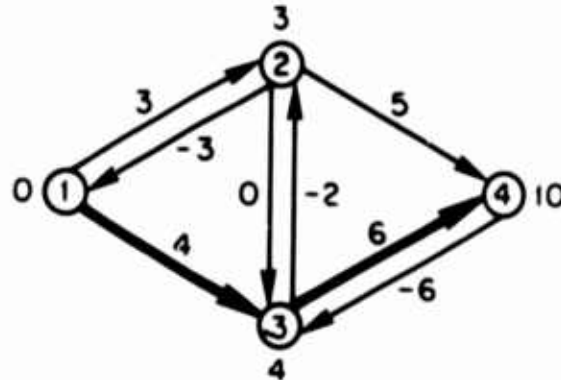


Fig. 5.4

directed cycles, such cycles have nonpositive lengths and consequently we can easily compute a longest chain from source to sink by an iterative procedure analogous to that of Sec. 2 for shortest chains in networks having nonnegative directed cycle lengths. This has been done for the example in Fig. 5.4, where the node potentials pick out the longest chain 1, 3, 4, shown in heavy arcs. This chain corresponds to a flow augmenting path, and we increase the flow along this path as described below. First note that arcs of Fig. 5.4 which have the same orientation as arcs of the project network correspond to possible forward arcs of the desired flow augmenting path, whereas arcs having opposite orientations correspond to possible reverse arcs of the flow augmenting path. Consequently if flow in an arc of the project network is to be increased, we treat c_{ij} as

a capacity if (i,j) is in state (a) or (b), and take infinite capacity for arcs in states (c) or (d). On the other hand, if flow in arc (i,j) is to be decreased, we take zero as a lower bound if (i,j) is in states (b) or (c), and take c_{ij} as a lower bound if (i,j) is in state (d). This means in the example that we send one unit of flow along the chain 1, 3, 4, thereby obtaining the flow shown in Fig. 5.5.

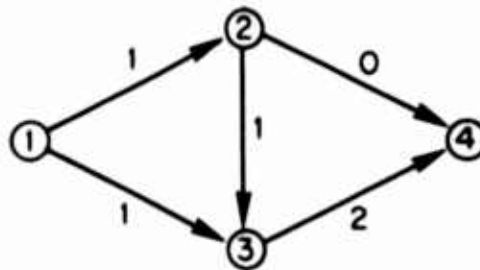


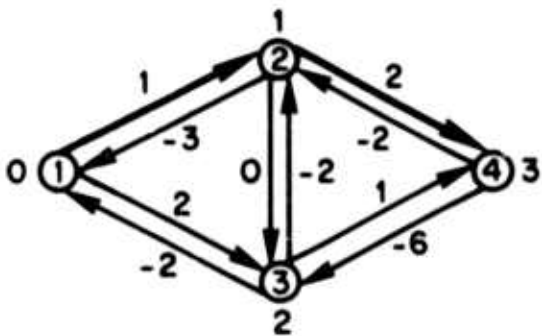
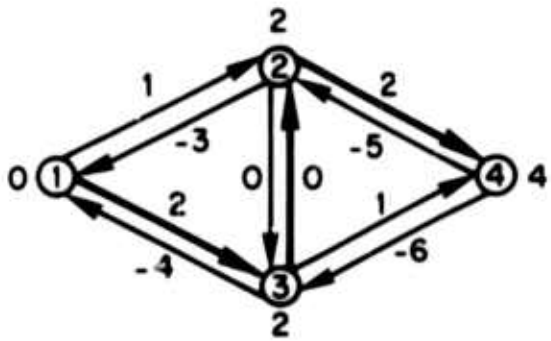
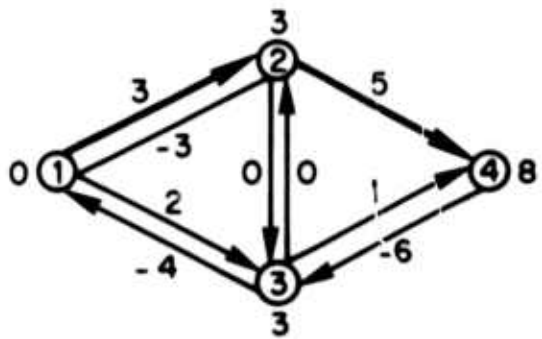
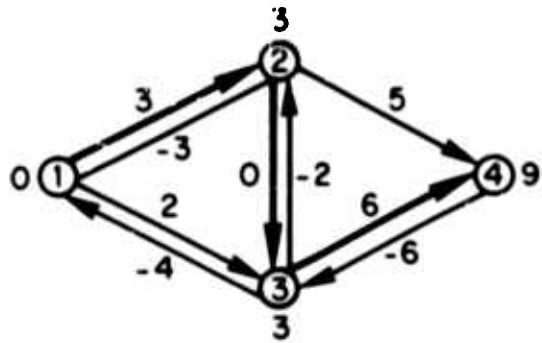
Fig. 5.5

The potentials of Fig. 5.4 constitute optimal event times corresponding to $\lambda = t_4 = 10$. Optimal job times are given by

$$(5.7) \quad t_{ij} = \min(b_{ij}, t_j - t_i) .$$

The procedure outlined above is then repeated, using the flow shown in Fig. 5.5. We obtain successively the results shown in Fig. 5.6 below.

Longest Chain Network



Flow Network

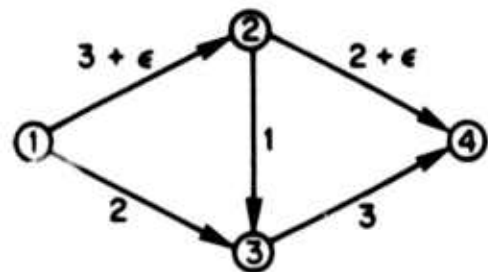
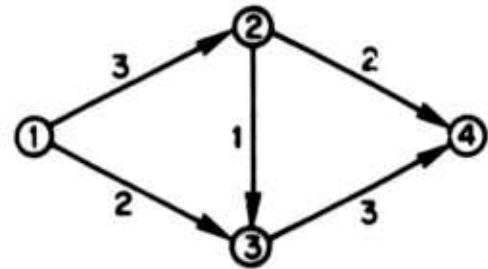
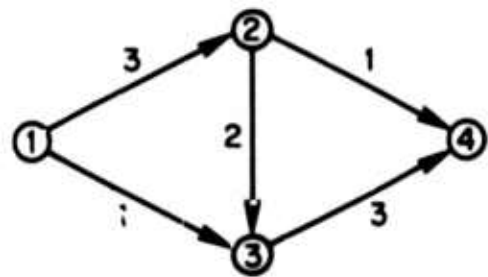
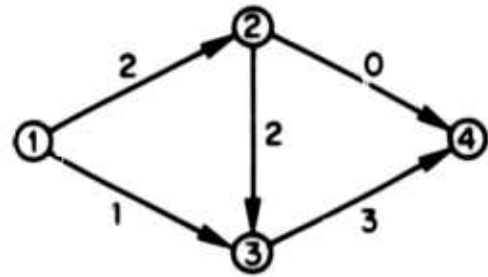


Fig. 5.6

In the last flow network of Fig. 5.6, there is no bound on the flow change ϵ . This signals termination of the computation. The complete minimum cost curve $P(\lambda)$ for the project has now been determined over the range of feasible λ , $3 \leq \lambda \leq 11$, the function $P(\lambda)$ being piecewise linear, with breakpoints at the successive values of $\lambda = t_4$ generated in the computation. Optimal event times for values of λ between two successive breakpoints are given by taking appropriate convex combinations of those corresponding to the two breakpoints, and optimal job times are then determined from (5.7).

Note that optimal times for a job (i,j) are not necessarily monotone in λ . For instance, the optimal job times for $(2,3)$ in the example are

$$2, 1, 0, 0, 0, 0, 1$$

corresponding to λ -values of

$$11, 10, 9, 8, 7, 4, 3$$

respectively. In other words, compressing the project time optimally can increase certain job times.

The method of this section can also be applied if job costs are piecewise linear and convex between crash and normal completion times. Such a job cost merely introduces other breakpoints in the function shown in Fig. 5.1, and the solution process is changed only in details.

REFERENCES

1. Busacker, R. G., and P. J. Gowen, "A Procedure for Determining a Family of Minimal Cost Network Flow Patterns," O.R.O. Technical Paper 15, 1961.
2. Dantzig, G. B., and D. R. Fulkerson, "Minimizing the Number of Tankers to Meet a Fixed Schedule," Nav. Res. Log. Q., Vol. 1, 1954, pp. 217-222.
3. ———, "On the Max-Flow Min-Cut Theorem of Networks," Linear Inequalities and Related Systems, Annals of Math. Study 38, Princeton Univ. Press, Princeton, New Jersey, 1956, pp. 215-221.
4. Dilworth, R. P., "A Decomposition Theorem for Partially Ordered Sets," Annals of Math., Vol. 51, 1950, pp. 161-166.
5. Ford, L. R., Jr., Network Flow Theory, The RAND Corporation P-923, 1956.
6. Ford, L. R., Jr., and D. R. Fulkerson, "Maximal Flow Through a Network," Can. J. Math., Vol. 8, 1956, pp. 399-404.
7. ———, "A Simple Algorithm for Finding Maximal Network Flow and an Application to the Hitchcock Problem," Can. J. Math., Vol. 9, 1957, pp. 210-218.
8. ———, "Constructing Maximal Dynamic Flows from Static Flows," Op. Res., Vol. 6, 1958, pp. 419-433.
9. ———, Flows in Networks, Princeton Univ. Press, Princeton, New Jersey, 1962.
10. Fulkerson, D. R., "A Network Flow Computation for Project Cost Curves," Man. Sci., Vol. 7, 1961, pp. 167-178.
11. ———, "Expected Critical Path Lengths in PERT Networks," Op. Res., Vol. 10, 1962, pp. 808-817.
12. Hu, T. C., "Parallel Sequencing and Assembly Line Problems," Op. Res., Vol. 9, 1961, pp. 841-849.
13. Jewell, W. S., "Optimal Flow Through Networks with Gains," Proc. Second International Conference on Operations Research, Aix-en-Provence, France, 1960.
14. Johnson, S. M., "Optimal Two- and Three-Stage Production Schedules with Setup Times Included," Nav. Res. Log. Q., Vol. 1, 1954, pp. 61-68.

15. Kelley, J. E., Jr., and M. R. Walker, "Critical Path Planning and Scheduling," Proc. Eastern Joint Computer Conference, Boston, Mass., 1959.
16. Kelley, J. E., "Critical Path Planning and Scheduling: Mathematical Basis," Op. Res., Vol. 9, 1961, pp. 296-321.
17. MacCrimmon, K. R., and C. A. Ryavec, "An Analytical Study of the PERT Assumptions," Op. Res., Vol. 12, 1964, pp. 16-38.
18. Malcolm, D. G., J. H. Roseboom, C. E. Clark, and W. Fazar, "Application of a Technique for Research and Development Program Evaluation," Op. Res., Vol. 7, 1959, pp. 646-669.
19. Ryser, H. J., Combinatorial Mathematics, Carus Math. Monograph No. 14, John Wiley and Sons, Inc., New York, 1963.
20. Tompkins, C. B., "Discrete Problems and Computers," I.N.A. - S3-5, Nov. 17, 1952.
21. Van Slyke, R. M., "Monte Carlo Methods and the PERT Problem," Op. Res., Vol. 11, 1963, pp. 839-860.